Scientific Journal of Impact Factor (SJIF): 5.71

e-ISSN (O): 2348-4470
p-ISSN (P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 7, Issue 11, November -2020

Using Text Classification Techniques to identify anti-patterns in SQL queries.

Dibyanshu Chatterjee[1], Niveditha CA[2]

[1]*Department of computer science and engineering, Ms Ramaiah University of Applied Sciences, Bangalore*
[2]*Application Software Analyst, Accenture, Bangalore*

Abstract — *A significant issue with utilizing social information bases, is formulating and writing proficient SQL queries. Some basic blunders known as anti-patterns are quite common in SQL queries and can genuinely affect its execution time and at times, the database's general execution. This paper manifests machine learning methods and identifies anti-patterns by approaching the issue as a text classification issue. Our outcome is a model dependent on a convolutional neural network that can be utilized to order a SQL query into zero, one or numerous anti-pattern classes.*

Keywords- *Anti-patterns, SQL, text classification techniques, relational database*

I. INTRODUCTION

With the expanding measure of data put away in social relational databases, it is important to compose SQL queries that execute quicker. Anti-pattern in SQL are basic errors that whenever kept away from, can make a query execute quicker. In this paper, we approach the issue as a multi-class multi-mark characterization issue. Our answer is pattern free, implying that the choice made by the neural network doesn't rely upon the database's legitimate structure. The dataset utilized has been worked from SQL queries given by Sky-Server from Sloan Digital Sky Survey (SDSS).

SkyServer, the entryway from the SDSS inventory, gives information access apparatuses to stargazers and is used in logical training. Through SkyServer, clients can utilize the SQL language to query the Sloan Digital Server database. Since 2001, the entryway has seen in excess of 280 million SQL queries submitted by clients and those queries have been opened to general society through the diverse information discharges. We bring 1 million queries from SkyServer, that we channel, measure and change. The last dataset of utilization contains 363616 remarkable SELECT queries.

Following a directed learning approach, the SQL inquiries from SkyServer are utilized as information; we physically mark the information by partnering each SQL query with a rundown of anti-patterns it contains.
Our model depends on a convolutional neural network prepared to classify a query into numerous classifications. We utilize the one-hot encoding procedure to encode the queries as word vectors. For encoding the anti-pattern classes we utilize a one dimensional tensor with each class spoke to as an integer.
We investigate a portion of the significant work in the field of SQL anti-pattern discovery in segment 5. In segment 2, we clarify in subtleties the cycle followed to construct the dataset. Then, we examine our model architecture in segment 3. In segment 4, we dissect the outcomes from our analyses. At long last in the end, we contrast our work with the current arrangements and investigate the conceivable future work.

II. DATASET OPERATIONS

II.1 Query collection: We begin constructing our dataset, by getting 1 million effective SQL queries from the SkyServer list:

```
        SELECT TOP 1000000 s t a t e m e n t
        FROM S q l L o g
   WHERE e r r o r = 0
```

A portion of these queries should be sifted through(filtered), to manufacture a more engaged dataset.

II.2 Filtration process: From the brought queries, we eliminate the duplicates, so the dataset genuine queries as it were.

```
a l l Q u e r i e s = l i s t ( s e t ( v a l u e s ) )
```

And then begins the removal of all the queires without “SELECT”

```
i m p o r t r e
a l l Q u e r i e s = l i s t (
f i l t e r (
l a m b d a i t e m : r e . s e a r c h (
” ˆ s e l e c t ” ,
i t e m . l o w e r ( )
) ,
a l l Q u e r i e s
)
)
```

III.3 Transformation: To kill unimportant data and diminish the size of our dataset jargon, we replace the entirety of the schema related terms contained in the queries with standard words. In this way the query contain practically just standard SQL keywords.

SELECT name f r o m s t u d e n t s ;
will be transformed to
SELECT column f r o m t a b l e ;

III. MODELLING

III.1 Encoding of Data:
III.1.1 Queries: Each SQL query in our current dataset is a rundown of words. Word portrayal strategies generally fall into two classifications. The main comprises of techniques, for example, one-hot vectors. This strategy is hazardous because of homonymy and polysemy words. The other class comprises of utilizing unregulated learning technique to acquire constant word vector portrayals. Latest examination results have shown that nonstop word representations are all the more impressive.

In this paper, we use word embedding dependent on word2vec. To encode the SQL query of our dataset, we decide to utilize the pre-prepared google word2vec embedding. The model is prepared on 100 billion words from Google News by utilizing the Skip-gram strategy and maximizing the normal log likelihood of the apparent multitude of words utilizing a softmax function. Our outcome model contains 123.852 tokens.

III.1.2 working with Anti-patterns: As our work depends on 16 Fixed anti-patterns, we encode the marked information as 1D Vector of whole numbers.

III.2 More into convolutional neural network: The convolution neural network is a best in class technique to demonstrate semantic portrayals of sentences. The convolution activity has been commonly used to integrate lexical ngram information. In our model, we utilize three distinctive convolutional channels with changing convolution window size to shape equal CNNs so they can get familiar with various kind of encoding of nearby locales in order to supplement each other to improve model precision and correctness. The last yield is the concatenation of the yield of each.

IV. CONDUCTED EXPERIMENTS

IV.1 Settings: For every one of our analyses, we utilize the Stochastic Gradient Descent optimization algorithm with a learning pace of 0.1 and a weight rot of 0.95. We lead the investigations with 50 epochs and we utilize mini batches of size 64. We assess the model each 100 stages. We use google pre-prepared word2vec accordingly the element of each word vector is 300.

We study the sensivity of the proposed model to the convolutional locale size, the quantity of convolutional highlight and the dropout rate. We found that we accomplish the best presentation when we utilize the settings esteems recorded in the Table I.

Region size	Feature Maps	Dropout rate
(4, 5, 6)	150	0.4

Table I

IV.2 Verification process: For approving our model we utilize the iterated K-Fold approval model.
The dataset is part into 10 smaller than usual datasets, which are utilized to approve every subset consistently.

IV.3 Outcomes: We contrast our outcomes and Sqlcheck. Sqlcheck is a build up device that depends on syntax checking rationale, to identify anti-patterns in SQL queries. We run a SQL on each of our dataset query, and store the outcomes, which we at that point contrast with our CNN results.

In the wake of running the examinations, our model can identify anti-pattern in a query with an exactness of 83.2.

V. RELAVANT WORKS

Normal mix-ups in SQL has been as of now in light of a legitimate concern for scientists before the presence of the ISO SQL-92 norm. In 1985, Welty concentrated how human elements can influence clients in utilizing SQL and found that client execution could be altogether improved. Afterward, Brass et al. begun taking a shot at the programmed discovery of legitimate blunders in SQL queries and broadened their work with the acknowledgment of basic semantic errors.

The actualized SQLLint apparatus which had the option to consequently distinguish these mistakes in (linguistically right) SQL statements. The instrument is by all accounts unsupported today. There is another online apparatus named SQLLint, however it is a SQL beautifier.

There are likewise books here. The Art of SQL and Refactoring SQL Applications favorable to vide rules to compose productive queries, while the book of Bill Karwin gathers antipatterns that ought to be evaded.

In a paper, Ahadi et al.,presented a huge scope investigation of understudies semantic slip-ups recorded as a hard copy SQL SELECT explanations. They gathered information from more than 2,300 understudies across nine years and summed up normal missteps of the understudies. They found that a large portion of the missteps were made in inquiries which require a JOIN, a subquery or a GROUP BY administrator. We contend that inquiries typ-ically utilize more mind boggling grammar by and by com-pared to understudy ventures. Henceforth, the circumstance can be surprisingly more terrible.

In the domain of implanted SQL, Christensen et al. proposed a strategy and an instrument (JSA, Java String Analyzer) to separate string articulations from Java code statically. As a possible use of their methodology, they check the grammar of dynamically created SQL strings. They limit their approach to the syntactic approval of the inquiries.

Wassermann et al. propose a static string investigation strategy to recognize potential mistakes in progressively created SQL code. With the usage of a CFL-reachability calculation they distinguish type blunders (e.g., linking a character to a number worth). Their methodology works with extricated inquiry strings of legitimate SQL linguistic structure. In a device demo paper, they present their model device called JDBC Checker.

As of late, Anderson and Hills considered question development designs in PHP. They broke down inquiry strings installed in PHP code with the assistance of the PHP AiR structure.

Quality appraisal of implanted SQL was supportive of presented by Brink et al. in 2007. They examined inserted question strings in PL/SQL, Cobol, and Visual Basic projects while they propose a nonexclusive methodology which could be applied to Java as well. They research connections which could be recognized through implanted inquiries (e.g., access, duplication, control conditions) and they propose quantitative inquiry measures for quality evaluation.

Numerous static strategies which attempt to manage inserted question strings do it with the end goal of SQL infusion recognition. Yeole and Meshram distributed a review of these strategies. SQL infusion discovery is diverse as the objective is explicitly to decide if a question could be influenced by client input.

A few papers likewise tackle SQL deficiency confinement methods. A powerful methodology was proposed by Clark et al. to confine SQL shortcomings in information base applications. They give order SQL tuples to show the SQL explanations executed at information base communication focuses.

An ongoing work of Delplanque et al. focuses on the information base to evaluate the nature of the composition and to recognize configuration smells in it. They execute an apparatus called DBCritics which can investigate PostgreSQL construction dumps and recognize configuration scents, for example, missing essential keys or unfamiliar key references.

An instrument which likewise must be referenced here is the Eclipse module called Alvor and JSA [17], this module investigates the string articulations in Java code. Furthermore, Alvor checks sentence structure right ness, semantics rightness, and article accessibility by contrasting the removed questions against its internal SQL syntax and by checking SQL statements against a genuine information base.

VI. CONCLUSION AND FUTURE IMPROVEMENTS

In this work, we tested utilizing text order methods to distinguish enemies of examples in SQL inquiries. The model uses a neural organization with a custom dataset worked from SkyServer list SQL questions. Exploratory outcomes show that, our model is very precise and can outflank build up punctuation checking programming.

For the future, we could zero in on reworking queries dependent on the anti-pattern distinguished.

VII. APPENDIX

Anti-pattern clarification select *

At the point when you SELECT *, you're regularly recovering a bigger number of sections from the database than your application actually needs to work. This makes more information move from the information base worker to the customer, easing back access and expanding load on your mama chines, just as setting aside more effort to traverse the organization.

Consider a situation where you need to tune a query to a significant level of execution. If you somehow happened to utilize, and it returned a larger number of sections than you really required, the worker would regularly need to perform more costly techniques to recover your information than it in any case may.

At the point when you SELECT *, it's conceivable to recover two sections of similar name from two distinct tables. This can frequently crash your data consumer null usage.

Null isn't equivalent to zero. A number ten more noteworthy than an obscure is as yet an obscure. Null isn't equivalent to a line of zero length. Consolidating any string with NULL in standard SQL brings NULL back. Null isn't equivalent to bogus. Boolean articulations with AND, OR, and NOT likewise produce results that a few people find confounding not invalid utilization.

At the point when we proclaim a segment as NOT NULL, it ought to be on the grounds that it would look bad for the line to exist without an incentive in that section.

String connection

You may need to drive a section or articulation to be non-invalid for improving the question rationale, yet you don't need that incentive to be put away. Use COALESCE capacity to develop the con-catenated articulation with the goal that an invalid esteemed segment doesn't cause the entire articulation to get invalid.

Bunch by use

Each segment in the select-rundown of an inquiry must have a solitary worth line for every line gathering.

Request by rand utilization

Arranging by a nondeterministic articulation (RAND()) implies the arranging can't profit by a file
Design coordinating use

VII. REFRENCES

[1] Poonyanuch Khumnin and Twittie Senivongse. 2017. SQL antipatterns detection and database refactoring process. 2017 18th IEEE/ACIS International Con-ference on Software Engineering, Artificial Intel-ligence, Networking and Parallel/Distributed Com- puting (SNPD)

[2] Csaba Nagy and Anthony Cleve. 2017. A Static Code Smell Detector for SQL Queries Embedded in Java Code. PReCISE Research Center, University of Na-mur, Belgium

[3] Natalia Arzamasova, Martin Schler, and Klemens Bhm. 2018. Cleaning Antipatterns in an SQL Query Log. IEEE Transactions on Knowledge and Data Engineering (Volume: 30 , Issue: 3)

[4] Bill Karwin 2010. SQL Antipatterns Avoiding the Pitfalls of Database Programming. The Pragmatic Bookshelf

[5] William J. Brown, Raphael C. Malveau, Hays W. Mc- Cormick III, and Thomas J. Mowbray. 1998. An-tipatterns. Wiley and Sons, Inc., New York.

[6] V. Singh et al. 2006 SkyServer Traffic ReportThe First Five Years Microsoft Research

[7] M. Jordan Raddick et al. 2014 Ten Years of SkyServer Tracking Web and SQL e-Science Usage Comput-ing in Science and Engineering, vol. 16(4)

[8] QFix: Di- agnosing errors through query histories eprint arXiv:1601.07539

[9] S. Brass et al. 2006 Semantic errors in SQL queries: A quite complete list Journal of Systems and Software, vol. 79, no. 5

[10] D. Burleson, V. Tropashko 2007 SQL Design Patterns: Expert Guide to SQL Programming Rampant Tech- Press, 2007